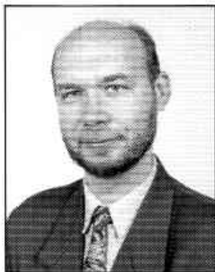
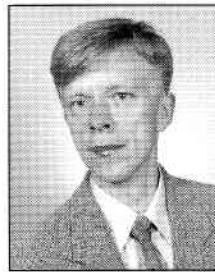
Cezary ZIELIŃSKI, Wojciech SZYNKIEWICZ

INSTYTUT AUTOMATYKI I INFORMATYKI STOSOWANEJ
POLITECHNIKA WARSZAWSKA

Sterownik robota POLYCRANK*



Dr hab. inż. Cezary ZIELIŃSKI
uzyskał stopnie: magistra inżyniera w 1982 r., doktora nauk technicznych w 1988 r., doktora habilitowanego w 1996 r. – wszystkie w Politechnice Warszawskiej. Obecnie pracuje tam na stanowisku profesora nadzwyczajnego i jest kierownikiem Zespołu Sterowania i Programowania Robotów w Instytucie Automatyki i Informatyki Stosowanej. Zajmuje się zagadnieniami związanymi z programowaniem, metodami sterowania oraz sterownikami systemów wielorobotowych wyposażonych w różnorodne czujniki. Wykłada: robotykę, układy logiczne oraz programowanie. Jest autorem i współautorem ponad 60 publikacji z dziedziny robotyki.



Dr inż. Wojciech SZYNKIEWICZ
uzyskał stopień magistra inżyniera w 1985 r., zaś doktora w 1996 r. na kierunku automatyka i robotyka na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. W Instytucie Automatyki i Informatyki Stosowanej Politechniki Warszawskiej pracuje od 1985 r., w tym od 1996 r. na stanowisku adiunkta. Zainteresowania naukowe obejmują problematykę sterowania i planowania działań robotów, układy wielorobotowe i współpracę robotów, systemy czasu rzeczywistego i ich zastosowania w robotyce. Jest autorem i współautorem około 30 prac poświęconych tej tematyce.

Streszczenie

Przedstawiono sterownik dla systemów składających się z wielu robotów, urządzeń współpracujących oraz czujników. Artykuł przede wszystkim koncentruje się na warstwie programowej sterownika. Prototypowy robot POLYCRANK traktowany jest jedynie jako jeden z obiektów sterowania.

Abstract

The paper presents a controller for systems containing: robots, cooperating devices and diverse sensors. The programmer tailors the controller exactly to match the task at hand. If the hardware configuration of the task at hand. If the hardware configuration of the system changes the programmer has the possibility of modifyng the software to accomodate the change. The paper focuses on the software component of the controller. In this case the prototype robot - named POLYCRANK - is treated as one of the components of the system and so the designed controller still has the potential of driving many robots.

Wstęp

Sterowniki robotów oraz języki programowania są nierozerwalnie związane ze sobą. System robotyczny zazwyczaj składa się z pewnej liczby robotów, urządzeń współpracujących oraz czujników. Zarówno liczba, jak i typ poszczególnych elementów systemu nie są znane przed zdefiniowaniem zadania lub klasy zadań, do realizacji których dany system ma być wykorzystywany. Przy konstruowaniu sterownika takiego systemu oraz określaniu sposobu jego programowania trzeba wziąć pod uwagę ten fakt. Istnieją dwa możliwe podejścia do rozwiązania powyższego problemu. Przy pierwszym podejściu sterownik ma strukturę zamkniętą (niemodyfikowalną), a metoda jego programowania musi umożliwiać określenie sposobu realizacji szerokiej klasy zadań. Sprowadza się to do zdefiniowania specjalizowanego języka programowania robotów wykonywanego przez sterownik o strukturze zamkniętej. Wtedy w skład sterownika wchodzi interpreter rozkazów języka specjalizowanego. Przy drugim podejściu sterownik może być dedykowany specyficznej konfiguracji sprzętowej systemu oraz niewielkiej klasie zadań, ale wtedy musi istnieć szybka i łatwa metoda projektowania takich specjalizowanych sterowników. Wówczas program użytkowy stanowi integralną część sterownika, więc zbędny jest interpreter. Pociąga to za sobą wykorzystanie uniwersalnego języka programowania oraz dodanie biblioteki modułów programowych obsługujących różne konfiguracje sprzętowe oraz realizujące różne zadania. Taki dedykowany sterownik konstruowany jest przez złożenie go z tych modułów. Ponieważ do biblioteki można ciągle dodawać nowe moduły, to mówimy, że sterowniki z nich konstruowane mają strukturę otwartą. Wraz ze zmianą zadania lub konfiguracji sprzętowej zmienia się sterownik - rekonstruuje się go używając nowych modułów albo modyfikując stare. W tym artykule przedstawiono właśnie to podejście.

Historycznie, sterowniki zamknięte oraz ich specjalizowane języki programowania pojawiły się wpierw. W latach siedemdziesiątych i osiemdziesiątych to podejście dominowało (np.: WAVE [12], AL [10], AML [15], RAPT [1], SRL [4], TORBOL [16]). Wkrótce jednak okazało się, że taki sterownik musi interpretować bardzo złożony język. W praktyce możliwości takiego języka były równorzędne uniwersalnym językom programowania komputerów. Ponadto trzeba było dodać instrukcje związane ze sterowaniem robotami oraz wykorzystaniem informacji z czujników. Nawet jeżeli punktem wyjścia do definicji specjalizowanego języka programowania robotów był język uniwersalny, to należało zdecydować o jakie specyficzne dla systemów robotycznych elementy trzeba go rozszerzyć. Rozwiązanie tego problemu nie było oczywiste, ponieważ konfiguracja sprzętowa systemu może się znacznie różnić w zależności od zadania. Dlatego też podejście zakładające konstrukcję zamkniętego sterownika zostało zarzucone w przypadkach, w których spodziewano się znacznych modyfikacji konfiguracji sprzętowych. Paradoksalnie okazało się, że sterowniki zamknięte, ze swej istoty mające wykonywać szeroką gamę zadań, są dużo lepiej przystosowane do obsługi systemów dedykowanych jedynie pewnym klasom zadań. W takim przypadku język może zostać przykrojony do potrzeb zarówno konfiguracji systemu jak i klasy zadań, które ma realizować. Im klasa jest szersza, tym bardziej uniwersalny musi być język, a więc i bardziej skomplikowany interpreter tego języka. Wkrótce jednak pojawił się nowy pomysł. Zamiast definiowania języka, który miałby prawie wszystkie możliwości uniwersalnego języka programowania oraz pewne dodatki związane ze specyfiką sterowania robotami, łatwiej jest wykorzystać któryś z istniejących języków programowania komputerów oraz dodać ele-

* *Praca została zrealizowana w ramach prac statutowych Politechniki Warszawskiej*

menty do obsługi robotów w postaci biblioteki modułów programowych. To podejście zastosowano w: PASRO [3, 4], RCCL [6], ARCL [5], RCI [9], KALI [7, 8, 2], RORC [18, 17], MRROC [18, 19]. Zazwyczaj korzystano z podejścia proceduralnego do tworzenia bibliotek, ale obecnie zmienia się to na korzyść podejścia obiektowego [20]. Ponadto, na drodze rozważań teoretycznych wykazano, że liczba modułów w bibliotece może być ograniczona do bardzo rozsądnej wielkości [21÷24]. Te same badania wykazały, że niezależnie od liczby i typu zarówno robotów jak i czujników, uniwersalny język programowania komputerów musi być rozszerzony albo tylko o jedną, ale za to złożoną, albo o dwie, ale prostsze instrukcje ruchu, aby stać się językiem programowania robotów zdolnym do sterowania dowolnym systemem robotycznym. To podejście zostało wykorzystane do konstrukcji sterownika systemu jednorobotowego, którego głównym elementem jest robot POLYCRANK [11]. Język służący do konstruowania tego typu sterowników został nazwany MRROC++ (*Multi-Robot Research-Oriented Controller*). W przypadku MRROC++ pojęcia: język, biblioteka, sterownik oraz system mogą być używane wymiennie, aczkolwiek nie są synonimami. Wynika to z faktu, że MRROC++ w zależności od fazy projektowania pełni funkcję każdego z wyżej wymienionych pojęć. Ponieważ językiem uniwersalnym leżącym u podłoża MRROC++} jest język C++, do nazwy dodano dwa plusy. Pomimo że aktualnie sterownik nadzoruje pojedynczego robota, to zastosowano MRROC++, czyli bibliotekę przeznaczoną do konstruowania sterowników dla systemów wielorobotowych zawierających dowolne urządzenia współpracujące oraz wyposażonych w różnorodne czujniki. Uczyniono tak, by w przyszłości nie zamykać drogi dla dołączenia kolejnych robotów do systemu.

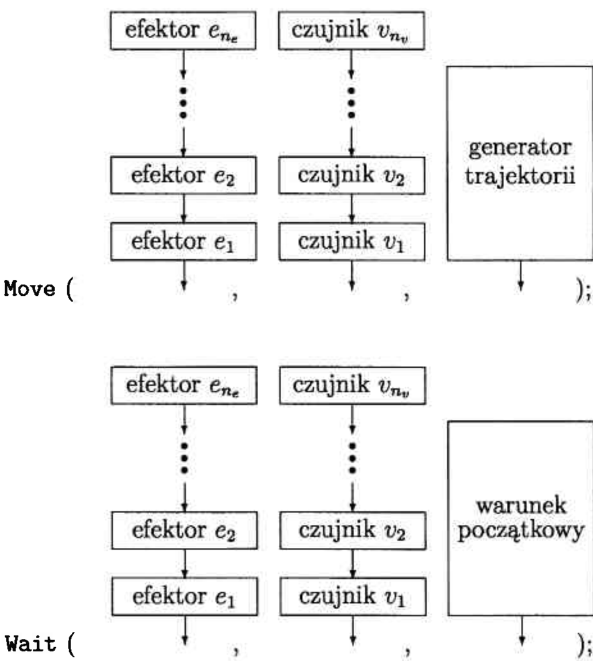
Struktura sterownika

W każdym systemie robotycznym można wyróżnić trzy części:

- **efektory** – czyli te elementy systemu, które oddziałują na otoczenie, zmieniając jego stan (mogące przemieszczać obiekty i narzędzia) - mogą to być roboty, ale również np. podajniki lub obrabiarki,
- **receptory** – czujniki rzeczywiste (sprzętowe) zbierające informacje o stanie otoczenia, np. kamery, czujniki zbliżeniowe, dalmierze, czujniki sił i momentów sił,
- **podsystem sterujący** – składający się zarówno ze sprzętu obliczeniowego (komputera lub sieci komputerów) jak i jego oprogramowania.

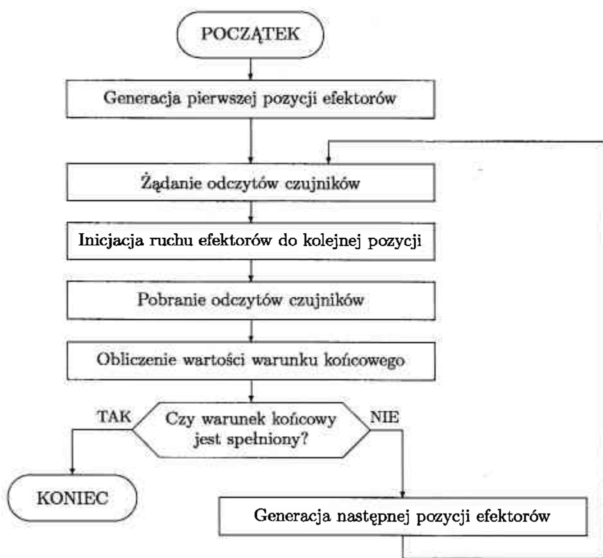
Ponieważ czujniki rzeczywiste rzadko dostarczają informacji w postaci bezpośrednio przydatnej do sterowania ruchem, więc dane pomiarowe z kilku prostych czujników albo z jednego złożonego muszą ulegać agregacji, czyli ekstrakcji informacji użytecznej. Ta zagregowana informacja stanowi odczyt czujnika wirtualnego.

Czujniki wirtualne mogą być wykorzystane albo do monitorowania aktualnego stanu systemu i jego otoczenia albo do sterowania, czyli do wpływania na przyszłe stany tych dwóch elementów. Monitorowanie może być użyte do stwierdzenia, czy spełnione są warunki rozpoczęcia ruchu, wtedy mówimy o monitorowaniu warunku wstępnego lub początkowego. Może też być wykorzystane przy określaniu warunku zakończenia ruchu, wówczas mamy do czynienia z monitorowaniem warunku końcowego - odbywa się to w trakcie trwania ruchu. Ostatnim przypadkiem, do którego przydatne jest monitorowanie, jest wykrywanie sytuacji awaryjnych. Język MRROC++ umożliwia użycie czujników do wszystkich trzech typów monitorowania jak i sterowania systemem. Do sterowania ruchem efektorów, przy wykorzystaniu czujników, służą instrukcje ruchu. Jak wspomniano we wstępie, można zdefiniować jedną złożoną instrukcję ruchu, która obejmie wszystkie możliwe przypadki sterowania [21÷24]. Niemniej jednak, dla zachowania prostoty zapisu, zdecydowano się na wprowadzenie dwóch oddzielnych instrukcji (rys. 1). Pierwszą jest Move, która odpowiedzialna jest za sterowanie ruchem oraz monitorowanie

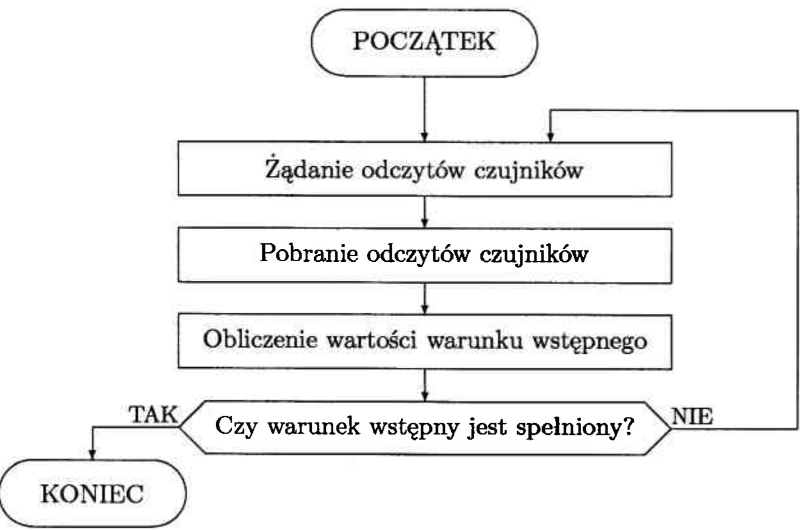


Rys. 1. Instrukcje ruchu systemu MRROC++

warunku końcowego (rys. 2). Drugą jest instrukcja Wait, realizująca monitorowanie warunku początkowego (rys. 3). Monitorowanie awarii zorganizowane zostało jako obsługa wyjątków, a więc odbywa się niejako równolegle do wykonania powyższych instrukcji, jak też dowolnych innych instrukcji programu sterującego. Obie z instrukcji korzystają z dwu list



Rys. 2. Sieć działań instrukcji Move



Rys. 3. Sieć działań instrukcji Wait

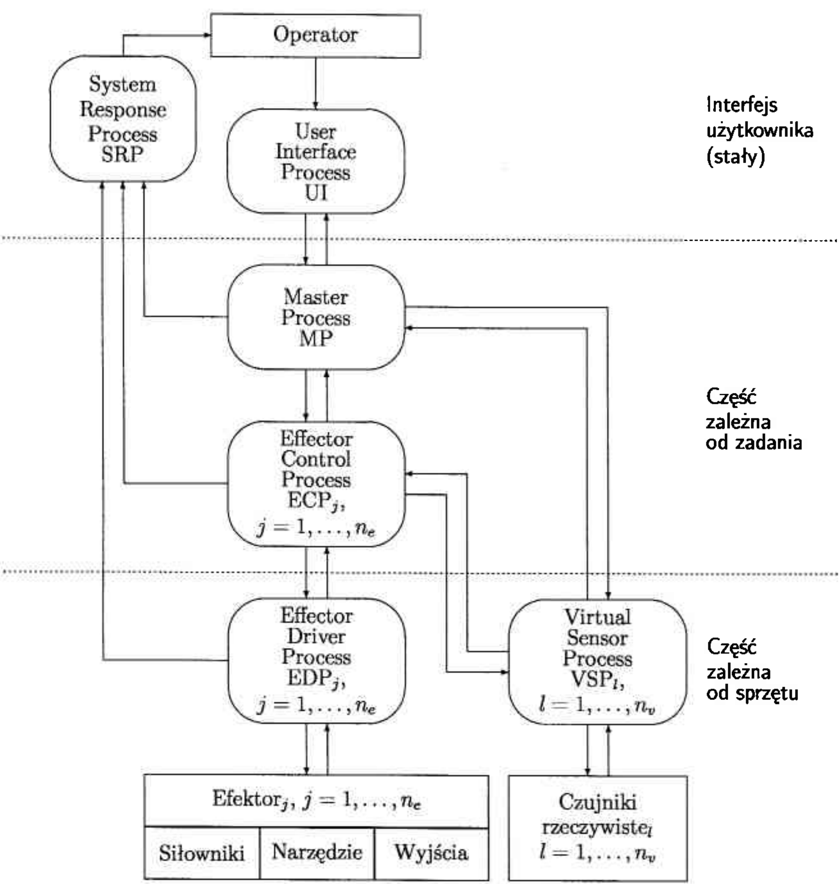
obiektów: robotów (albo ogólniej efektorów) oraz czujników wirtualnych. Ponadto instrukcja Move wykorzystuje obiekt zwany generatorem trajektorii. Obiekt ten określa zarówno warunek końcowy jak i sposób sterowania ruchem efektorów.

Natomiast instrukcja Wait wymaga dostarczenia obiektu, który definiuje warunek początkowy. Oba te obiekty stanowią porcję kodu w języku C++ dostarczaną przez użytkownika. Zależy ona od rodzaju zadania, które system ma wykonać.

Zgodnie z koncepcją tworzenia układów sterujących za pomocą języka MRROC++, sterowniki systemów wielorobotowych mają hierarchiczną strukturę funkcjonalną. Poszczególne funkcje utworzonego sterownika realizowane są przez moduły w postaci odrębnych procesów działających w węzłach sieci lokalnej. Modularność struktury sprawia, że wymiana jednego z elementów systemu nie pociąga za sobą zmiany pozostałych. Ogólna struktura funkcjonalna tego typu sterowników przedstawiona została na rys. 4.

Można wyróżnić kilka warstw w układzie sterowania. Warstwa najniższa odwołuje się bezpośrednio do sprzętu. W jej skład wchodzą procesy obsługi urządzeń (ang. drivers) (efektorów, czujników). Procesy EDP (ang. Effector Driver Processes) odpowiedzialne są za rozwiązanie prostego i odwrotnego zadania kinematyki oraz bezpośredni dostęp do sprzętu, a więc za funkcje charakterystyczne dla danego typu robota, a nie za realizację zadania. Procesy VSP (ang. Virtual Sensor Processes) realizują odczyt i przetwarzanie danych z czujników rzeczywistych. Warstwa druga - procesy ECP (ang. Effector Control Processes) - realizuje algorytmy sterowania poszczególnymi efektorami adekwatnie do wykonywanego aktualnie przez system zadania. Warstwa nadrzędna - proces MP (ang. Master Process}) - odpowiada za koordynację procesów sterujących efektorami. Warstwy, w skład których wchodzą procesy MP i ECP, zależne są jedynie od zadania, które ma być zrealizowane przez system. Program użytkowy stanowiący zapis sposobu realizacji postawionego przed systemem zadania umieszczony jest wewnątrz procesów MP i ECP. Dodatkowo istnieje jeszcze warstwa nie związana bezpośrednio z realizacją zadania - procesy UI (ang. User Interface) i SRP (ang. System Response Process) zapewniające komunikację sterownika z operatorem systemu.

Warstwa komunikacji użytkownika z rozproszonym sterownikiem wielorobotowym składa się z dwóch modułów: procesu User Interface obsługującego zlecenia operatora oraz procesu System Response Process wyświetlającego komunikaty o stanie systemu. Oba procesy realizują komunikację z użytkownikiem za pośrednictwem okienkowego środowiska graficznego QNX Windows 4.2. Użytkownik musi zadbać o zapewnienie prawidłowych mechanizmów komunikacji pomiędzy napisanym przez siebie procesem sterującym Master Process, a procesem User Interface. Mechanizmy te pozwalają na rozpoczęcie oraz przerwanie wykonania procesu sterującego efektorami. Proces UI obsługuje zlecenia operatora systemu i odpowiada za inicjację oraz zakończenie dzia-



Rys. 4. Struktura sterownika MRROC++ (ne - liczba efektorów, nv - liczba czujników wirtualnych)

łania sterownika wielorobotowego. Użytkownik może sterować ręcznie robotami za pomocą okienkowego menu, poruszając się w ramach skończonej listy dostępnych zleceń. Zakres dostępnych zleceń zależny jest od bieżącego stanu systemu. Lista zleceń obejmuje: ładowanie i usuwanie procesów EDP i MP, uruchamianie i zakończenie oraz wstrzymywanie i wznawianie wykonania procesu MP, obsługa ruchów ręcznych (synchronizacja robota, ręczne ruchy na poziomie: położeń wałów silników, współrzędnych wewnętrznych oraz zewnętrznych), zakończenie działania systemu.

Proces SRP odbiera komunikaty od wszystkich procesów, które informują użytkownika o zaistnieniu szczególnej sytuacji (zmiana stanu systemu, informacja o błędzie), formatuje i wyświetla komunikaty na ekranie monitora, przechowuje w buforze nadchodzące komunikaty, tak aby można było śledzić zachowanie systemu od momentu jego uruchomienia.

Proces MP składa się z dwu podstawowych części: stałej powłoki (niemodyfikowalnej części procesu) oraz części modyfikowanej przez użytkownika. W części stałej procesu dokonuje się: rejestracja procesu MP w systemie operacyjnym, powoływanie procesów potomnych oraz inicjacja kanałów komunikacyjnych między odpowiednimi procesami. Struktura oraz funkcje części modyfikowanej procesu MP, zależą od rodzaju oraz złożoności zadania.

W zależności od rodzaju zadania i liczebności efektorów programista umieszcza instrukcje Move i Wait oraz inne polecenia pomocnicze (instrukcje C++) zarówno w części zmiennej procesu MP jak i procesów ECPj.

Komunikacja procesu ECPj z procesami VSPl, których on używa, może być zrealizowana jako przekaz:
- interaktywny lub
- nieinteraktywny.

W przypadku komunikacji interaktywnej ECPj wysyła żądania danych do swoich VSPl. Procesy VSPl odczytują adekwatne czujniki rzeczywiste, dokonują agregacji danych i przesyłają uzyskany w ten sposób odczyt czujnika wirtualnego z powrotem do ECPj. W międzyczasie ECPj może sterować efektorem. Jeżeli odczyty nie nadejdą do chwili gdy ECPj wykona operację odczytu danych, jest on zawieszany do chwili ich nadejścia.

W przypadku komunikacji VSPl wyzwalany jest przerwaniami. Gdy czasomierz wygeneruje przerwanie, VSPl odczytuje adekwatne czujniki rzeczywiste, dokonuje agregacji danych i otrzymany wynik wstawia do bufora. Następnie proces VSPl jest zawieszany. W ten sposób każdy z procesów ECPj może odczytać ostatnie dane w każdej chwili. Oczywiście dostęp procesów do bufora musi być synchronizowany (wzajemne wykluczanie).

Dla architektury sprzętowej składającej się z lokalnej sieci komputerów klasy PC, jednym z najlepiej dostosowanych do potrzeb implementacji złożonych struktur sterowania jest rozproszony system wielozadaniowy czasu rzeczywistego QNX4.2x [13, 25]. System ten składa się z egzekutora wielozadaniowego i zespołu zadań (procesów) systemowych, współpracujących ze sobą i zadaniami użytkowymi zgodnie z modelem wymiany usług (ang. client-server model). Odwołania zadań użytkowych do zadań systemowych mają postać wiadomości, przekazujących podczas spotkania żądanie wykonania określonego zlecenia. Możliwość realizacji spotkania między dowolnymi zadaniami w sieci sprawia, że każde zadanie może korzystać z dowolnych zasobów całej sieci lokalnej. Modularna struktura i brak prywatnych połączeń między zadaniami użytkowymi umożliwiają elastyczne konfigurowanie systemu stosownie do potrzeb poszczególnych aplikacji.

Każde dołączenie robota nowego typu do systemu pociąga za sobą stworzenie dla niego procesu EDPj. Ogólna struktura tego procesu została zdefiniowana w systemie MRROC++. Dalej zostanie przedstawiony zarówno sposób komunikacji z procesem EDPj jak i podsystem sterujący silnikami robota POLYCRANK.

PROCES EDP

Proces EDPj spełnia rolę interpretera poleceń przysyłanych z ECPj lub UI. Listę poleceń dla procesu EDP przedstawiono na rys. 5.

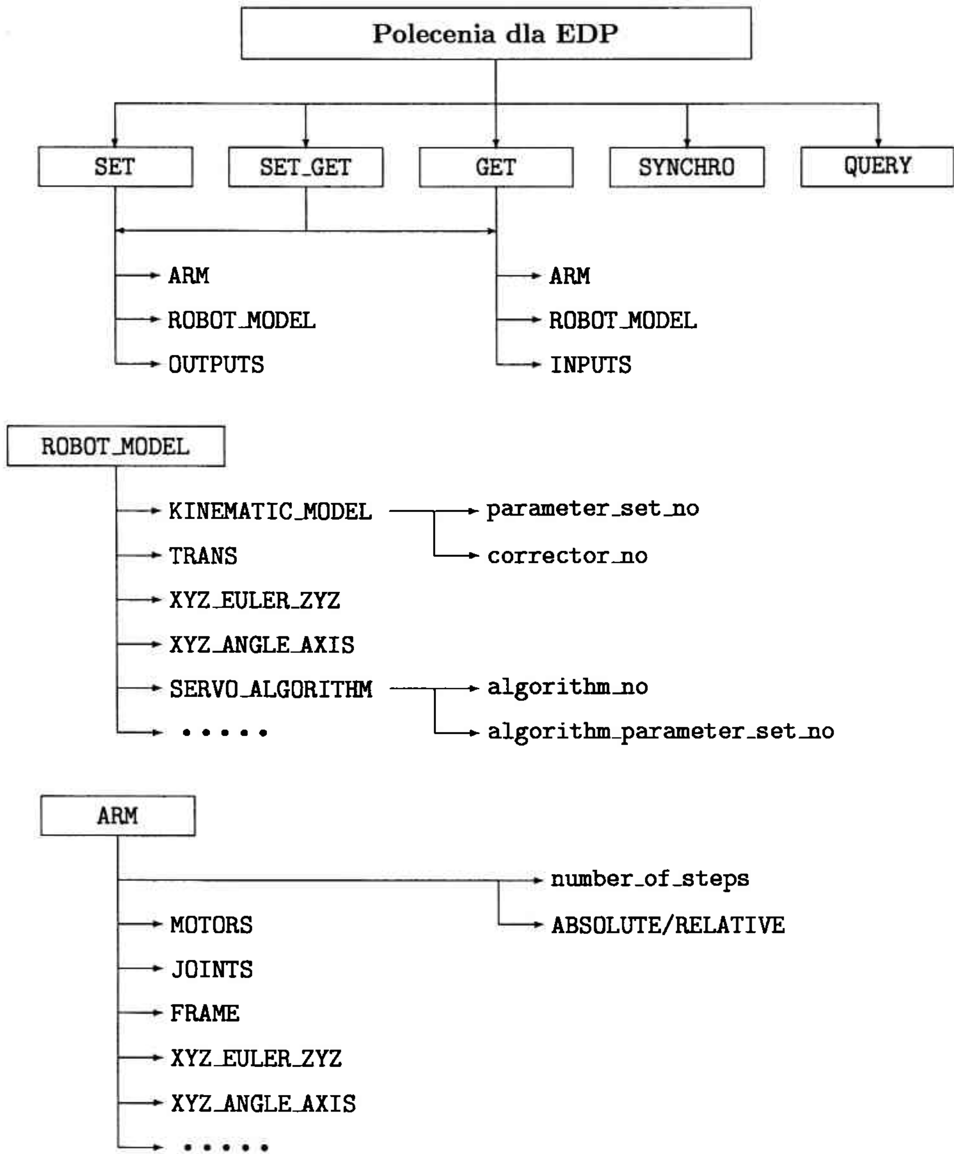
Komunikacja z procesem EDP (driver'em robota) zawsze składa się z dwóch faz. W fazie pierwszej informuje się go co ma być zrobione. Wtedy przesyłany jest jeden z rozkazów: SYNCHRO, SET, GET, SET_GET wraz z adekwatnymi parametrami. W fazie drugiej proces EDP jest pytany o rezultat wykonania uprzednio wysłanego rozkazu za pomocą polecenia QUERY. Dopiero wtedy EDP przyśle dane, o które był pytany lub jeśli nie żądano żadnych danych, potwierdzi wykonanie polecenia albo poinformuje o zaistniałym błędzie. Proces EDP zawsze zachowuje się biernie - stanowi server dla klienta (procesów ECP i UI).

Po uruchomieniu procesu EDP można mu wysłać jedno z dwóch poleceń: SYNCHRO lub SET. Przed synchronizacją ro-bota rozkaz ruchu czyli zlecenie SET ARM dostępne jest tylko w jednej postaci. Można jedynie zlecać ruch względny zadawany w postaci przyrostów położenia wałów silników SET ARM MOTORS RELATIVE). Inne formy rozkazu SET ARM są niedopuszczalne w tym stanie i powodują błąd. Spowodowane jest to przyrostowym pomiarem położenia wałów silników, a więc koniecznością inicjacji liczników położenia po włączeniu sterownika. Wysłanie rozkazu SYNCHRO spowoduje przemieszczenie wszystkich osi robota do położenia czujników synchronizacji. Synchronizacja wykonywana jest tylko na początku działania systemu. Może być ewentualnie wykonywana powtórnie po wystąpieniu błędu fatalnego, który powoduje utratę synchronizacji, wówczas proces EDP wraca do stanu początkowego, takiego jak jest tuż po uruchomieniu i inicjacji procesu EDP. Informacja o sposobie zakończenia synchronizacji (bezbłędne lub z błędem) przekazywana jest procesowi ECP po wydaniu przez niego polecenia QUERY. Jeśli robot jest zsynchronizowany, powtórny rozkaz SYNCHRO nie będzie wykonany zostanie jedynie przesłany komunikat z błędem ALREADY_SYNCHRONISED.

Po bezbłędnym wykonaniu synchronizacji do dyspozycji procesu ECP są tylko rozkazy SET, GET, SET_GET (we wszystkich wariantach) - w pierwszej fazie komunikacji z EDP; oraz rozkaz QUERY - w drugiej fazie.

Polecenie SET może dotyczyć:
- zmiany parametrów geometrycznych narzędzia aktualnie zamontowanego na manipulatorze,
- ustawienia wyjść binarnych układu sterującego,
- zmiany zbioru parametrów modelu kinematycznego robota,
- zmiany korektora kinematycznego,
- zmiany algorytmów regulacji,
- zmiany parametrów (nastaw) serwomechanizmów osi,
- przemieszczenie ramienia robota.

Polecenie GET powoduje:
- odczytanie parametrów geometrycznych narzędzia aktualnie zamontowanego na manipulatorze,
- odczytanie stanu wejść binarnych układu sterującego,
- odczytanie aktualnego numeru algorytmu regulacji,
- odczytanie numeru aktualnego zbioru parametrów serwomechanizmów,
- odczytanie aktualnego położenia ramienia robota.



Rys. 5. Lista poleceń dla EDP

Zarówno w przypadku polecenia SET jak i GET można spowodować każdą z wymienionych czynności osobno, wszystkie razem albo dowolną ich kombinację, w zależności od użycia parametrów rozkazu.

Polecenie SET_GET jest superpozycją poleceń SET i GET.

Przykładowo za jego pomocą można zlecić ustawienie wyjść, odczytanie wejść, przemieszczenie ramienia oraz odczytanie jego położenia po zakończeniu ruchu. Oczywiście dane zwrotne zostaną przekazane dopiero po wydaniu polecenia QUERY. Położenie i orientacja narzędzia (TOOL) względem kołnierza manipulatora oraz końcówka ramienia (ARM) względem globalnego układu odniesienia mogą być określone na różne sposoby m.in.:

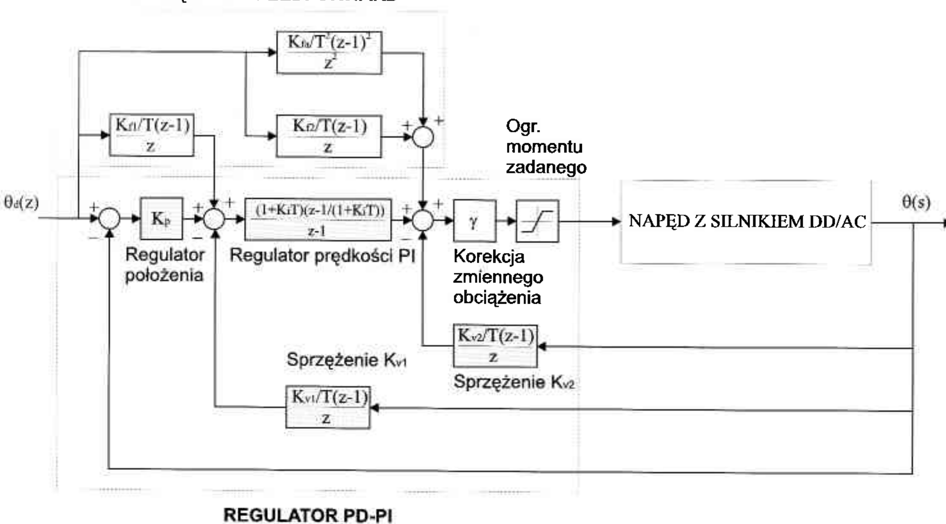
- jako macierz reprezentująca trójścian związany z końcówką,
- jako trzy współrzędne kartezjańskie oraz trzy kąty Eulera (ZYZ) trójścianu związanego z końcówką,
- jako trzy współrzędne kartezjańskie oraz wektor stanowiący oś obrotu trójścianu związanego z końcówką (długość tego wektora jest proporcjonalna do kąta obrotu).

Ponadto położenie ramienia (ARM) może być określane przez podanie współrzędnych wewnętrznych łańcucha kinematycznego lub przez zadanie położeń wałów silników elektrycznych.

Odczyt położenia robota wykonywany jest jedynie we współrzędnych bezwzględnych, natomiast ruch może być zrealizowany we współrzędnych bezwzględnych lub względnych (przyrostowo w stosunku do aktualnej pozycji). Każde polecenie ruchu (SET ARM) traktowane jest jako makrokrok. Dodatkowym parametrem polecenia jest liczba kroków, w której ma być wykonany makrokrok.

Współpraca ze sprzętem

Do sterowania robotem POLYCRANK został zbudowany hierarchiczny system wielokomputerowy, złożony z sieci komputerów typu IBM-PC oraz komputera trójprocesorowego wykorzystującego magistralę VME [14]. W przypadku zadań niezbyt wymagających obliczeniowo sterownik może korzystać je-



Rys. 6. Struktura serwomechanizmu pojedynczej osi

dynie z komputera trójprocesorowego. Dodatkowe komputery są połączone z nim siecią Ethernet i są zarządzane przez system operacyjny czasu rzeczywistego QNX [13, 25]. W skład komputera trójprocesorowego wchodzą: procesor Intel 486 (PEP-1) oraz dwa procesory Motorola 68060 (PEP-2 i PEP-3). Ponieważ procesor Intel dysponuje zasobami odpowiadającymi standardowej konfiguracji komputera typu IBM-PC, może stanowić dla komputera trójprocesorowego bramę do komputerów połączonych siecią Ethernet i w związku z tym zarządza nim system operacyjny QNX. Procesory Motorola i Intel połączone są wspólną magistralą VME i komunikują się przez wspólną pamięć. Każdy z procesorów Motorola steruje trzema napędami ramienia robota. Jeden steruje silnikami bezpośredniego napędu DD (direct drive) poruszającymi trzema głównymi stopniami swobody, natomiast drugi steruje silnikami AC poruszającymi kiścią. Procesory Motorola pracują pod nadzorem systemu operacyjnego czasu rzeczywistego OS-9.

W związku z powyższą strukturą sprzętową proces EDP został podzielony na trzy podprocesy. Na procesorze Intel wykonywany jest podproces EDP_MASTER, który stanowi interpreter poleceń (rys. 5) zlecanych procesowi EDP przez procesy nadrzędne. Natomiast procesory Motorola wykonują podprocesy SERVO (tj. SERVODD i SERVOAC) odpowiedzialne za realizację serworegulacji dla dwóch trójek napędów. W przypadku niewielkiej obliczeniochłonności zadania, procesor Intel wykonuje w podziale czasu również pozostałe procesy (tj. MP, ECP, UI oraz SRP).

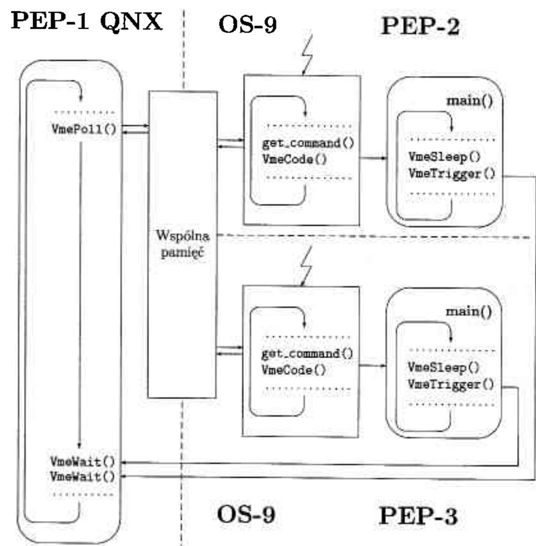
Zadaniem podprocesu EDP_MASTER jest:

- rozkodowanie rozkazu i sprawdzenie jego poprawności,
- wykonanie adekwatnych operacji matematycznych na parametrach polecenia (np. przeliczenie współrzędnych),
- wykonanie rozkazu (np.: zmiana definicji narzędzia, odczytanie wejść binarnych, zapisanie wyjść binarnych, odczyt aktualnego położenia ramienia)
- zlecenie wykonania ruchu, jeżeli przysłano rozkaz ruchu (SET ARM) lub synchronizacji (SYNCHRO),
- odebranie informacji o sposobie realizacji ruchu przez podprocesy SERVO,
- uformowanie przesyłki zwrotnej dla zleceniodawcy (ECP lub UI),
- wysłanie do ECP przesyłki zwrotnej w odpowiedzi na polecenie QUERY.

Podproces EDP_MASTER ma strukturę automatu skończonego, zmieniającego swój stan pod wpływem napły-wających rozkazów. Obsługę błędów zrealizowano jako reakcję na wyjątki zgłaszane w różnych częściach programu. Reakcja zawsze następuje na tym samym poziomie - tzn. na najwyższym poziomie podprocesu, niezależnie od miejsca gdzie wykryto błąd. Błąd jest analizowany i podejmowana jest decyzja, od którego stanu automat powinien wznowić działanie. Oczywiście zleceniodawca jest informowany o zaistniałej awarii.

Podprocesy SERVO odpowiedzialne są za realizację algorytmów regulacji dla wszystkich stopni swobody robota oraz za bezpośredni dostęp do sprzętu sterującego. W aktualnej wersji serwomechanizmów realizowany jest algorytm PI+PD z dodatkowym sprzężeniem tachometrycznym oraz sprzężeniami wyprzedzającymi (ang. feedforward) od prędkości i przyspieszenia zadanego (rys. 6). Czas powtarzania algorytmu wynosi 1 ms (dla wszystkich trzech osi) i dyktowany jest przerwaniem zegarowym. Uwzględniono możliwość bez zakłóceniowego przełączania algorytmów regulacji oraz zmiany ich parametrów.

Sposób komunikacji podprocesów EDP_MASTER i SERVO przedstawiono na rysunku 7. Komunikacja odbywa się przez wspólną pamięć procesora Intel. EDP_MASTER umieszcza rozkazy dla podprocesów SERVO we wspólnej pamięci wykorzystując funkcję VmePoll zapisaną w języku C. Funkcja ta powoduje jednocześnie odebranie danych o stanie procesów SERVO, więc i stanie napędów robota. Podprocesy SERVO odbierają z pamięci wspólnej rozkazy w swych procedurach obsługi przerwań. Czynią to z wykorzystaniem funkcji get_command zapisanej w języku C. Ponieważ uruchomienie pośrednika (proxy) w procedurze obsługi przerwania jest niedopu-



Rys. 7. Współpraca EDP_MASTER po stronie QNX z serworgulatorami po stronie OS-9

szczalne, procedura ta budzi program główny, wykonywany przez procesor Motorola, funkcją VmeCode zapisaną w języku C. Dopiero program główny uruchamia pośrednika (funkcja VmeTrigger), który przekazuje programowi wykonywanemu przez procesor Intel informację o odebraniu poprzedniego polecenia. Informacja ta jest sygnałem do sformowania i przekazania kolejnego polecenia przez {\rm EDP_MASTER}. Po uruchomieniu pośrednika procedura obsługi przerwania realizuje odebrane polecenie. Polecenie to może być realizowane wieloetapowo przy kolejnych przerwaniach. Jeżeli do czasu zrealizowania całości polecenia nie zostanie nadesłane nowe, to procedura obsługi przerwania przechodzi do realizacji kroków biernych, czyli wystawiania sterowania zachowującego osiągnięte położenie.

W obu procesorach PEP-2 i PEP-3 przerwania są generowane niezależnie, zaś synchronizacja działania procesów SERVODD i SERVOAC odbywa się poprzez EDP_MASTER (funkcje VmeWait). Częstotliwość generowania przerwań może być zmieniana. Całość obliczeń algorytmu regulacji jest wykonywana w procedurze obsługi przerwania.

Podsumowanie

Zarówno metoda projektowania sterowników, jak i konkretny sterownik dedykowany robotowi POLYCRANK, przechodzą obecnie intensywne próby. Skonstruowano kilka sterowników przeznaczonych do realizacji różnych zadań. Przy okazji stworzono interfejs z użytkownikiem umożliwiający uczenie robota albo wczytywanie pliku zawierającego współrzędne węzłów trajektorii. Plik ten może być wygenerowany automatycznie, np. przez program CAD albo stworzony dowolnym edytorem tekstów (oczywiście należy zachować odpowiedni format danych). Testy potwierdziły prawidłowość przyjętych założeń. Dzięki językowi/bibliotece/systemowi MRROC++ można konstruować sterowniki dedykowane różnym zadaniom oraz konfiguracjom sprzętowym. System MRROC++ steruje również dwoma innymi robotami. Są to: prototypowy robot RNT oraz robot IRp-6 posadowiony na torze jezdnym. Oczywiście sprzęt sterujący w obu przypadkach jest inny niż opisany powyżej, a w związku z tym podprocesy (SERVO) są inne. Ponieważ roboty te mają inną strukturę kinematyczną oraz inny sprzęt sterujący, więc również ich podprocesy EDP_MASTER są inne, aczkolwiek realizują te same funkcje.

BIBLIOGRAFIA

[1] A. P. AMBLER, D. F. CORNER: RAPT1 User's Manual. Department of Artificial Intelligence, University of Edinburgh, 1984.

[2] P. BACKES, S. HAYATI, V. HAYWARD, K. TSO: The KALI Multi-Arm Robot Programming and Control Environment. Proc. NASA Conf. on Space Telerobotics, 1989.

[3] C. BLUME , W. JAKOB: PASRO: Pascal for Robots. Springer-Verlag, Berlin 1985.

[4] C. BLUME, W. JAKOB: Programming Languages for Industrial Robots. Springer-Verlag, 1986.

[5] P. CORKE, R. KIRKHAM: The ARCL Robot Programming System. Proc. Int. Conf. Robots for Competitive Industries, Brisbane, Australia, 14-16 July 1993, str. 484-493.

[6] V. HAYWARD, R. P. PAUL: Robot Manipulator Control Under Unix RCCL: A Robot Control C Library. Int. J. Robotics Research, Vol. 5, No.4, Winter 1986, str.94-111.

[7] V. HAYWARD, S. HAYATI: KALI: An Environment for the Programming and Control of Cooperative Manipulators. Proc. American Control Conf., 1988, str. 473-478.

[8] HAYWARD V., DANESHMEND L., HAYATI S.: An Overview of KALI: A System to Program and Control Cooperative Manipulators. In: Advanced Robotics. Ed. Waldron K., Springer-Verlag, 1989, str. 547-558.

[9] J. LLOYD, M. PARKER, R. McClain: Extending the RCCL Programming Environment to Multiple Robots & Processors. Proc. of the IEEE Int. Conf. on Robotics & Automation, 1988, str. 465-469.

[10] S.MUJTABA, R. GOLDMAN: AL Users' Manual. Stanford Art. Int. Lab., 1979.

[11] K. NAZARCZUK, K. MIANOWSKI: POLYCRANK - Fast Robot without Joint Limits. Proc. of the 12th CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators, RoManSy'98, 6-8 July 1998, Springer Verlag, 1998. str. 317-324.

[12] R. PAUL: WAVE: A Model Based Language for Manipulator Control. The Industrial Robot, March 1977, str. 10-17.

[13] K. SACHA: Laboratorium systemu QNX. Oficyna Wydawnicza Politechniki Warszawskiej. Warszawa 1995.

[14] K. SACHA: Oprogramowanie systemowe wieloprocesorowego sterownika szybkiego robota. VI Krajowa Konferencja Robotyki, 9-12 wrzesień 1998, Świeradów Zdrój, Polska. Oficyna Wydawnicza Politechniki Wrocławskiej, 1998, tom 2, str. 279-286.

[15] R. H. TAYLOR, P. D. SUMMERS, J. M. MEYER: AML: A Manufacturing Language. Int. Journal of Robotics Research, Vol. 1, No. 3, 1982, str. 842-856.

[16] C. ZIELIŃSKI: TORBOL: An Object Level Robot Programming Language. Mechatronics, Vol.1, No. 4, Pergamon Press, 1991. str. 469-485.

[17] C. ZIELIŃSKI: Flexible Controller for Robots Equipped with Sensors. 9th Symp. Theory and Practice of Robots & Manipulators, Ro.Man.Sy'92, 1-4 Sept. 1992, Udine, Italy, Lect. Notes: Control \& Information Sciences 187, Springer-Verlag, 1993. str.205-214.

[18] C. ZIELIŃSKI: Robot Programming Methods. Publishing House of Warsaw University of Technology, 1995.

[19] C. ZIELIŃSKI: Control of a Multi-Robot System. 2nd Int. Symp. Methods & Models in Automation & Robotics MMAR'95, 30 Aug.-2 Sept. 1995, Międzyzdroje, Poland. str. 603-608.

[20] C. ZIELIŃSKI: Object-Oriented Robot Programming. Robotica, Vol.15, 1997. str. 41-48.

[21] C. ZIELIŃSKI: Object-Oriented Programming of Multi-Robot Systems Utilising Sensory Information. 3rd ECPD Int. Conf. on Advanced Robotics, Intelligent Automation and Active Systems, 15-17 September 1997, Bremen, Germany, str. 176-181.

[22] C. ZIELIŃSKI: Object-Oriented Programming of Multi-Robot Systems. 4th Int. Symp. Methods and Models in Automation and Robotics MMAR'96, 26—29 August 1997, Międzyzdroje, Poland, str. 1121-1126.

[23] C. ZIELIŃSKI, W. SZYNKIEWICZ: Systemy MRROC i MRROC++; Część I: Struktura. W: Konstrukcja, sterowanie i programowanie złożonych systemów robotycznych. Red.: C. ZIELIŃSKI, Zielińska T., Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 1997, str. 25-40.

[24] C. ZIELIŃSKI, W. SZYNKIEWICZ: Systemy MRROC i MRROC++; Część II: Realizacja. W: Konstrukcja, sterowanie i programowanie złożonych systemów robotycznych . Red.: C. ZIELIŃSKI, Zielińska T., Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 1997, str. 41-54.

[25] QNX System Architecture. Quantum Software, 1992.

Artykuł recenzowany